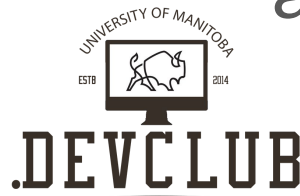


Introduction to Git Versioning with GitHub

Organized by University of Manitoba IEEE
and .devClub



Word From .devClub Executive



Mohsen Yousefian

Your UMIEEE Workshop Presenters



Mohsen Yousefian



Mahyar Mirrashed

Topics to be Covered (Agenda)

- Installing Git SCM on your personal machine
- Creating a GitHub account
- Git concepts (untracked files, staging area, and repositories)
- Git commands for creating projects
- Git commands for committing changes to projects
- Git commands for feature branching
- Git commands for feature merging

Approximated length: 50-60 minutes. Extra 10 minutes at end for questions.

Workshop Layout

- Workshop (mostly) aimed at professional users
 - Professional users use Git for workplace scenarios (follow along with shell)
 - Please use PowerShell if on Windows (Start Menu → [Search “PowerShell”])
 - Please use bash if on MacOS/Linux

Getting the Most From Online Workshops

- Take notes!
- Ask questions in Google Meet chat!
- Follow along if possible!

Take advantage of this UMIEEE workshop to the greatest extent possible!

Why Learn Git?

- Keep track of changes to multiple files
 - Allowing you to revert changes if necessary
- Simplifies working on files and projects with multiple people
 - Imagine working on PowerPoint presentation with multiple people using USBs

Installing Git SCM

Installing Git SCM

- For Windows: <https://git-scm.com/download/win>
 - If unsure regarding 32-bit and 64-bit distributions, head to Start Menu → About your PC → Device Specifications → System Type
- For Mac: <https://git-scm.com/download/mac>
 - May have to install Homebrew beforehand: <https://brew.sh/>
- For Linux: <https://git-scm.com/download/linux>

Head to shell and execute `git --version` to ensure proper functioning.

```
(base) x manzik@Manzik-MBP-2 ~/Documents/GitHub git --version  
git version 2.24.3 (Apple Git-128)
```

Creating GitHub Account and Code Repository

Creating GitHub Account

- Create GitHub account at <https://github.com/join>

Understanding: Code Repositories

- Similar to project folders, but for code
 - Contains all project files on cloud
 - Stores each file's revision history
- Repository types
 - Private individual repositories (only with GitHub Pro)
 - Publicly shared repositories (like open source projects)
- Every repository belongs to user account or team
 - Repository owners have complete control (deletion, management, etc.)


Understanding: Basic Git SCM Process

- Working area (untracked)
 - Modified files *not ready to commit* to repository
- Staging area (tracked)
 - Modified files *ready to commit* to repository
- Repository
 - Files *committed* to be stored on cloud



Creating a Git Repository

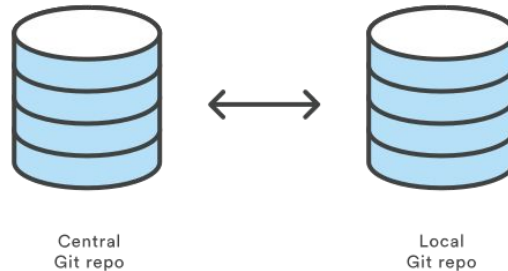
Creating your First Code Repository

- Head to GitHub main page at <https://github.com/>
 - Click on green “New” button → 
- Use following details during creation:
 - Repository name: Personal
 - Description: Personal repository for miscellaneous pieces of code.
 - Public/Private: [Select “Private”]
 - Add a README file: [Uncheck]
 - Add .gitignore: [Uncheck]
 - Choose a license: [Check]
 - For license type, select “MIT License”


Copying “Cloning”, Adding Files, and Making Commits on Git Repository

Understanding: Cloning

- Duplication is referred to as “cloning” in Git
 - **Origin/central** repository is like bank (single source of truth) on GitHub servers
 - Personal users duplicate/ “clone” from **origin** to obtain **local** version



Cloning an Origin

- Retrieve the link to your repository
 - On GitHub.com tab, select your repository from list
 - Head to repository page and click on “Code” button → 
 - Copy the HTTPS link
- On shell, using `cd`, move into parent folder for repository directory
 - Navigate (`cd`) to non-OneDrive synced directory (if on Windows) and execute `mkdir GitHub`
 - Afterwards, navigate into newly created directory
 - Execute `git clone <repository>`
 - `<repository>` will be same HTTPS link mentioned above
 - In case of (fatal) error, execute `git config --global core.askpass`

Creating Files in Local Repository

- Option 1: Use the graphical user interface
 - Right-click (on empty part of explorer) → New → Text Document
 - Name as “locations.txt”
 - Double-click file, type “Earth’s Moon”, and save document (CTRL + S)
- Option 2: Use the shell
 - Execute echo “Earth’s Moon” >> locations.txt
 - No response indicates successful file creation
 - Execute echo “Earth’s Moon” >> locations.txt on Linux or MacOS

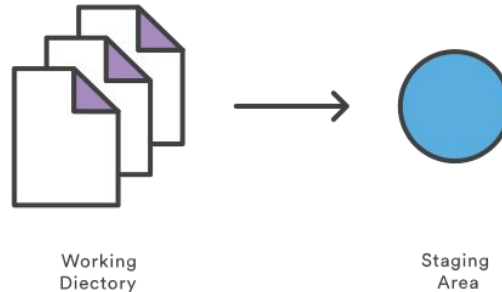
View Status of Local Repository (optional)

- On shell, execute `git status`
 - Shows that file is untracked (Git sees file not ever previously committed)

Untracked files are going to be **red**.

Adding Files to Local Repository

- On shell, instruct Git to track `locations.txt`
 - Execute `git add locations.txt`
 - No response indicates successful file creation
 - Command moves file from working directory to Git staging area
 - Used to prepare snapshot of set of changes prior to committing to history



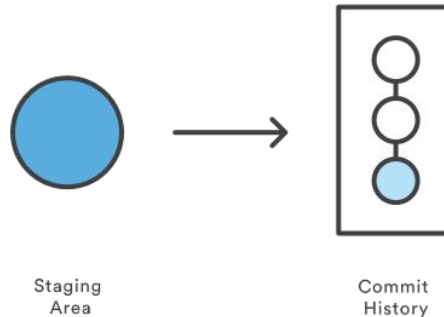
View Status of Local Repository (optional)

- On shell, execute `git status`
 - Shows added (staged) and tracked files
 - Changes are ready to be committed

Tracked files are going to be **green**.

Committing Files to Local Repository

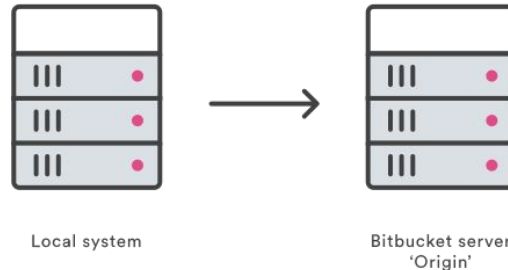
- On shell, execute `git commit -m "Initial commit"`
 - Takes staged "snapshot" and commits to project history
- Alongside `git add`, `git commit` defines basic workflow for most Git users



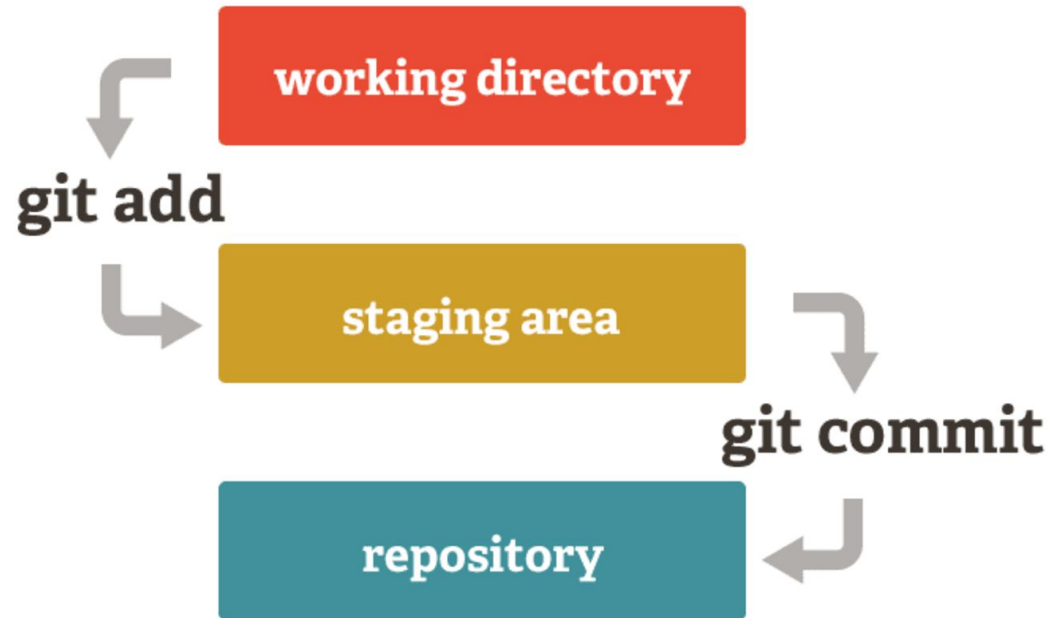
Pushing Files from Local to Origin Repository

- On shell, execute `git push`
 - Command specifies push to origin (central repository)

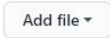

To check proper push, go to GitHub repository online, refresh page if needed, and view `locations.txt` as having been uploaded to origin.



Workflow Diagram for Local Changes



Creating Files on Origin Repository

- Head to created GitHub repository
- Click on white “Add file” button → 
 - Then, click “Create new file”
- For name field, enter “places.txt” in name field
 - Enter “Riding Mountain” in larger field portion
 - Scroll to bottom and click “Commit new file” → 

Pulling Files from Origin to Local Repository

- On shell, navigate to top level of local repository
 - Execute `git pull --all` to pull all changes from origin
 - Enter GitHub password if prompted

Pulling merges files from origin repository to local repository.

One last tip: Discarding local changes

- You can discard all of the changes you have made (revert working changes or not committed)
 - Execute `git checkout -- <filename(s)>` to discard your local working changes
 - Or `git checkout -- *` to discard all of the working changes

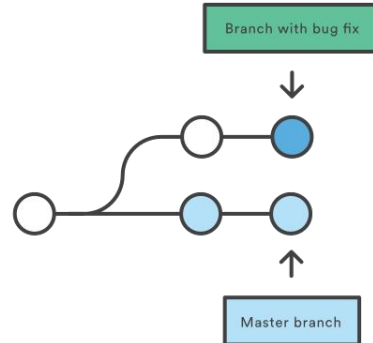
Understanding: Git-based Collaboration Workflow

- Communication with remote repositories (like GitHub) is foundation of every Git-based collaboration workflow
 - Every developer has copy of repository
 - With personal local history
 - With personal branch structure
 - Users share series of commits rather than single changesets
 - Rather than committing changesets, Git allows sharing of branches between repositories
- Manage connections with repositories
 - Publish local commits to the origin by “pushing”
 - Updating local files based on the origin by “pulling” commits

Branching and Merging Changes on Git Repository

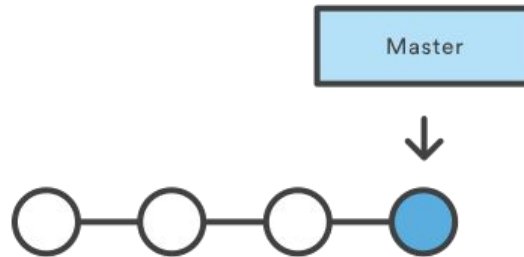
Understanding: Branches

- Allows file modification until ready (bug tested, standard verified, etc.)
- Represent independent lines of development
 - Features added without having everyone work on same file constantly
 - Can be “merged” when production ready
 - Git repositories automatically start with master (main) branch



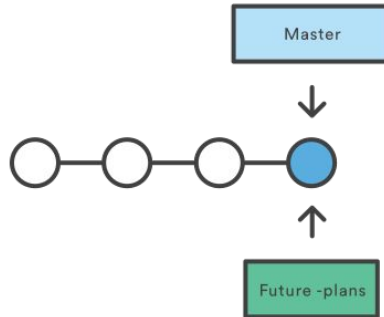
Understanding: Branches

- Simply commit pointers
 - Git creates new pointer rather than creating new set of files/folders
- Prior to creating branches, Git repository can be mapped as following:



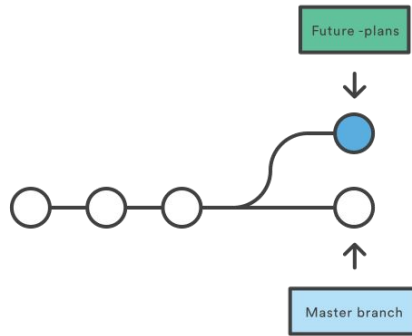
Adding Branch to Origin Repository

- To create a branch, head to top level of local repository
 - Execute `git branch future-parks` to create new branch named “future-parks”
 - Creates branch but does not switch you to that branch
- Repository at present state can be mapped as following:



Adding Branch to Origin Repository (continued)

- Pointer updated to current branch
- To work in new branch, “check out” to desired branch
 - Execute `git checkout future-plans`
- After checking out, repository can be mapped as following:



Creating Branch and Checking Out Shortcut

Use the following command to create a branch and check it out:

```
git checkout -b future-parks # create and checkout the branch
```

Which is same as the following command in a sequence.

```
git branch future-parks # create the branch
```

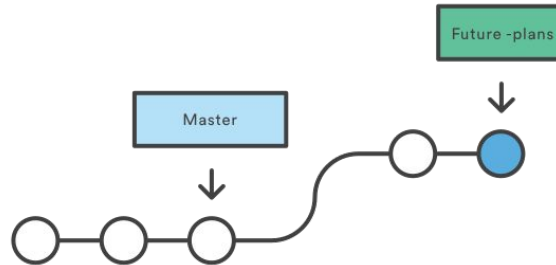
```
git checkout future-parks # checkout the branch
```

Using Newly Created Branch

- Now head to `locations.txt` and add another location: Yellowstone
 - Save file and close
 - Head to shell and execute `git status`
 - Notice that status indicates changed branch
- Stage (`git add`) and commit file (`git commit`)

Using Newly Created Branch (continued)

- With newest commit, Git repository can be mapped as following:
 - The future-parks branch is now ahead of the master branch



Merging via Fast-Forward Merging

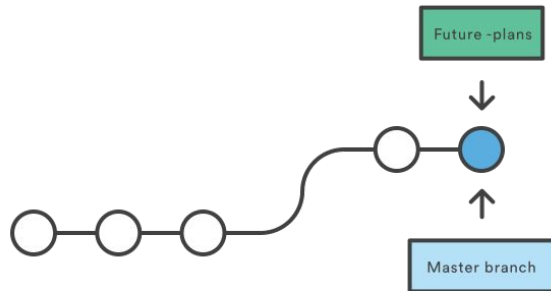
- After future-parks branch is developed enough, merge into master branch
 - *Enough* is vague term dependent on context
- *Fast-forward* merge because only one branch
 - Linear path exists from current branch tip to target branch
 - Combines current branch tip histories to target branch tip instead of “merging”
- Common for short-lived branches (not longer-running feature integrations)

Merging via Fast-Forward Merging (continued)

- Navigate to top-level of repository
- Verify changes are committed and current branch via `git status`
- Next switch to master branch via `git checkout`
 - Execute `git checkout master`
- Merge changes from future-parks to master branch
 - Execute `git merge future-parks`
- Delete future-parks afterward (not being planned to be used in the future)
 - Execute `git branch -d future-parks`
- See results of merge using `git status`

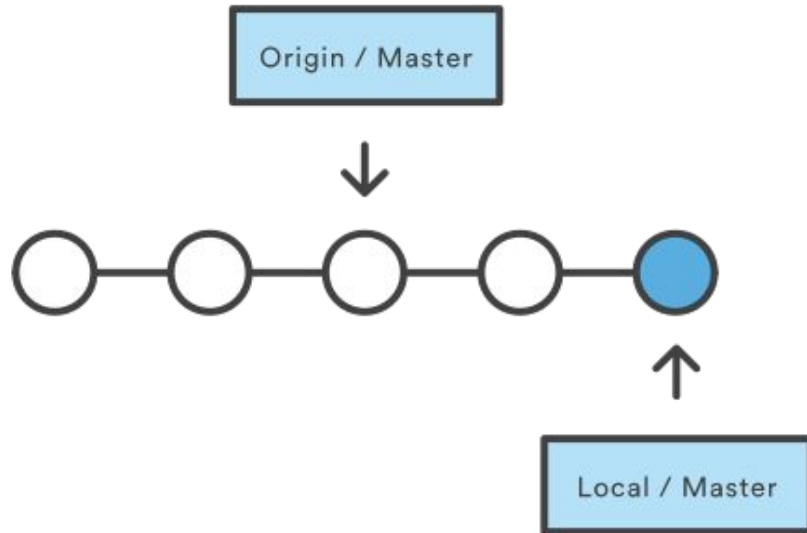
Merging via Fast-Forward Merging (continued)

- Effect represented as following:

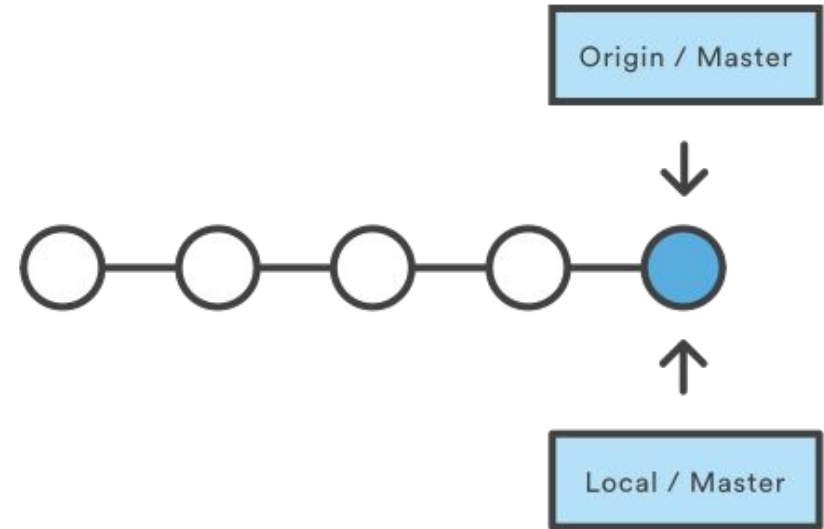


Sharing Local Changes to Origin

BEFORE PUSHING



AFTER PUSHING



Sharing Local Changes to Origin (continued)

- Push changes to remote repository
 - Execute `git push origin master`
 - Sometimes, `git push` also works (be careful, `git` will make assumptions for you!)

Other Notes (optional)

- University of Manitoba supplies students with GitHub Pro accounts
 - Head to <https://education.github.com/students> to claim Pro account
 - Will require student email address
 - Will take time to verify (Pro version not needed immediately for this workshop)
 - Head to <https://education.github.com/pack> to see additional University of Manitoba perks
- Alternative to command-line: GitHub Desktop (for casual users)
 - Use GitHub's GUI application to perform same operations as command line Git
 - Head to → <https://desktop.github.com/> to download (free)

Closing Remarks

Consider joining UMIEEE!

edu.ieee.org/ca-umieee/about-ieee/join-ieee-2/

Consider joining .devClub!

umdevclub.slack.com/signup

Questions?

Thanks for attending our workshop!

We will now take any additional remaining questions! (Please place in the chat.)

If you have more questions, join UMIEEE.